QPSK変調を利用した送信機の設計

――信号処理シミュレータとHDLシミュレータの協調シミュレーションで動作を確認

第3章

原田博司，横山明久

ここでは，ソフトウェア無線機の送信機の設計について解説する．変復調にはQPSK（quadrature phase shift keying）を用いる．QPSK送信機を作るために，PN符号生成器やフィルタなどの機能ごとに三つのブロックに分け，HDL化した．また，それぞれのブロックごとの評価やシステム・レベルの評価を行うためのツールの使用方法についても説明する．なお，ここで紹介した設計データは，本誌2005年1月号の付属FPGA基板上で動作させることができる．（編集部）

ここでは，第2章で述べた変復調方式の基礎をもとに，FPGAを用いたソフトウェア無線機を設計します．本稿では，とくにQPSK変調を用いた送信機を実現するためのFPGA設計，および基本的なツールの使用方法などを解説します．なお，ここではシステム・レベルのシミュレーションとHDL（hardware description language）レベルのシミュレーションを共存させるため，以下のツールを使用します．また，HDLの記述には，適当なテキスト・エディタを用いました．

- **システム・レベル・シミュレーション用ツール**
  MATLAB注1/Simulink（米国The MathWorks社製）
  System Generator for DSP（米国Xilinx社製）注2
- **HDLシミュレーション用ツール**
  ModelSim注3（米国Mentor Graphics社製）
- **論理合成，配置配線用ツール**
  ISE（Xilinx社製）注2

設計の流れは次のとおりです．

1）変調器の全体の構成を考えて，それをブロック化する
2）それぞれのブロックについて，HDLでソース・コードを記述する
3）そのソース・コードをSimulinkへロードする（System Generator for DSPが有するBlackbox機能を用いる）
4）ソース・コードを評価するためのモジュールをSimulink上で用意する
5）SimulinkとModelSimを協調させてシミュレーション（コシミュレーション）を行う
6）必要であれば，ほかのソース・コードを追加して，3）

注1：Version R13で動作を確認した．
注2：System Generator for DSPとISE Foundationの評価版は，本誌付属のDVD-ROMに収録されている．なお，筆者らは今回，論理合成，配置配線用ツールとしてISE WebPACK 6.3iを使用した．
注3：Version 5.8bで動作を確認した．なお，ModelSim XEは，本誌付属のDVD-ROMに収録されているISE Foundationの評価版に含まれている．

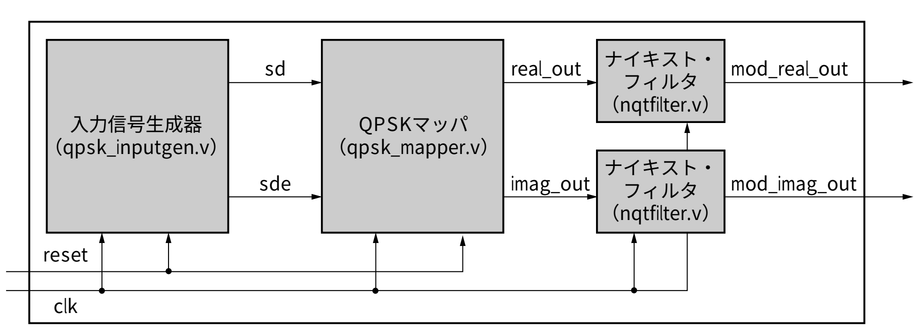


図1
設計する送信機の構成図
本特集で設計するQPSK変調処理システムの内部構成を示している．PN系列を生成する入力信号生成器，IQ信号を生成するQPSKマッパ，帯域制限処理を行うナイキスト・フィルタから構成される．

からの動作を繰り返す

7）すべて終了した後に，FPGAに対する論理合成，配置配線用のプロジェクト・ファイルを作成する

1 全体の構成を考えてブロック化する

では，QPSK変調を利用した送信機を設計していきます．

● 三つのブロックでシステムを構成する

まず，全体のブロック構成ですが，大きく分けると次の3種類のブロックからなります（**図1**）．

1）QPSK_INPUTGENブロック

2）QPSK_MAPPERブロック

3）NQTFILTERブロック

1）によって，QPSKの変調器に対する入力信号を生成します．ここでは疑似ランダム系列であるPN（pseudo noise）符号からなるランダム・データを作ることにします．入力はクロックclkとリセット信号reset，出力はランダム・データsdとsdデータに同期したイネーブル信号sdeです．

2）は，入力された信号sdとイネーブル信号sdeを受け，それらを**図2**に示すQPSK用の信号点に配置します．今回は2ビットの信号aとbを受信し，出力のデータreal_out（*I*チャネル）とimag_out（*Q*チャネル）を出力します．この信号は，後でフィルタをかけることを考慮して4クロックごとにデータを出力し，それ以外は‘0’データを出力します（詳細は本特集 第2章を参照）．

3）は第2章の**図9**（p.49）で説明したナイキスト・フィルタのブロックです．入力されるreal_out，imag_out，およびクロック信号clkを受け取り，**図3**に示すフィルタをかけます．これは，タップ数が17でロールオフ率αが1のナイキスト・フィルタです．このブロックから，フィルタ処理を行った変調信号mod_real_outとmod_imag_outが出力されます．

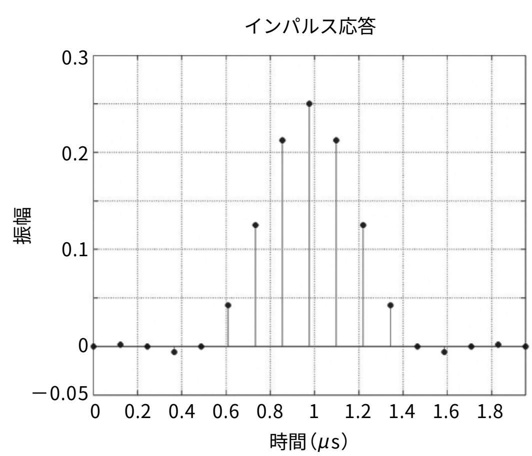


図3　ナイキスト型フィルタ

帯域制限を行うために利用するロールオフ率100%のナイキスト・フィルタのタップ係数値とインパルス応答値を示す．信号点間で影響を及ぼさないように，中心から4点離れた場所では係数が0となっていることがわかる．

2 ソース・コードを作成する

次に，各ブロックのソース・コードを作成します．

● QPSK_INPUTGENブロックでPN符号生成器を実現

まず，QPSK_INPUTGENブロックです．今回はHDLとしてVerilog HDLを用います．

リスト1は，**図4**に示すPN符号からなるランダム・データを生成するブロックのプログラムです．このPN符号生成器は，0〜6の番号が割り振られた七つのシフト・レジスタで構成されます．初期値として“1000000”が入力されていますが，クロックの入力に合わせて6番目のシフト・レジスタに入っている値が出力されます．このとき，6番目

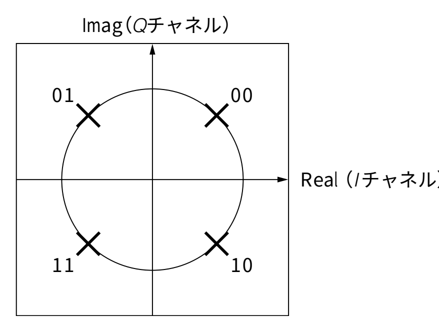


図2　QPSK変調における信号空間ダイヤグラム

QPSK変調において1信号点は2ビット入力信号により決定される．図に示すように，信号点の*x*軸側を*I*チャネル（Real），*y*軸側を*Q*チャネル（Imag）と呼ぶ．

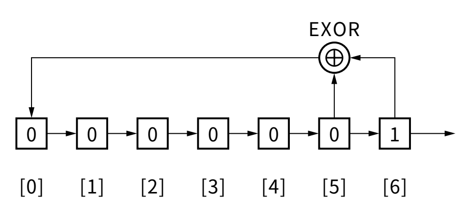


図4　PN符号器

7ビット・シフト・レジスタを利用して，[0] ビットについては [5] と [6] の排他的論理和を入れ，そのほかのビットについてはそれぞれ1ビットずつシフトさせる．出力は [6] ビットから取得する．初期化時に [6] ビットに‘1’を，そのほかのビットには‘0’を格納しておく．

3

リスト1　QPSK_INPUTGENブロック(qpsk_inputgen.v)

```
module qpsk_inputgen (clk, reset, sd, sde);
input clk;
input reset;
output sd;
output sde;

//********** Input Signal
wire          clk;
wire          reset;
//**********

//********** Output Signal
reg           sd;
reg           sde;
//**********

//********** Register
reg           counter;
reg     [6:0] shiftreg;
//**********

//********** Internal Signal
reg           exor_var;
//**********

always @(posedge clk) begin
  if (reset==1) begin
    counter = 1'b0;
    shiftreg = 7'b100_0000;

    sde = 1'b0;
  end
  else begin
    counter = ~counter;

    if (counter==1'b0) begin
      exor_var = shiftreg[5]^shiftreg[6];
      sd = shiftreg[6];
      shiftreg = {shiftreg[5:0],exor_var};
      sde = 1'b1;
    end
    else
      sde = 1'b0;
  end
end


endmodule
```

リスト2　QPSK_MAPPERブロック(qpsk_mapper.v)

```
module qpsk_mapper (clk, reset, rd, rde, real_out,
imag_out);

input         clk;
input         reset;
input         rd;
input         rde;
output  [7:0] real_out;
output  [7:0] imag_out;

//********** Input Signal
wire          clk;
wire          reset;
wire          rd;
wire          rde;
//**********

//********** Output Signal
reg     [7:0] real_out;
reg     [7:0] imag_out;
//**********

//********** Register
reg           counter;
reg           rd_buf;
//**********

//********** Internal Signal
wire    [1:0] rdbufandnow_var;
reg           counter_var;
//**********

always @(posedge clk) begin
  if (reset==1'b1) begin
    real_out     = 8'b0000_0000;
    imag_out     = 8'b0000_0000;
    counter      = 1'b0;
    rd_buf       = 1'b0;
  end
  else begin
    counter_var = counter;

    if (rde==1'b0) begin
      imag_out = 8'b0000_0000;
      real_out = 8'b0000_0000;
    end

    else if (counter_var==1'b0) begin
      counter_var = ~ counter_var;
      rd_buf = rd;
    end

    else begin
      counter_var = ~ counter_var;

      case ({rd_buf, rd})
        2'b00,2'b01 : imag_out = 8'b0111_1111;
        2'b10,2'b11 : imag_out = 8'b1000_0001;
      endcase

      case ({rd_buf, rd})
        2'b00,2'b10 : real_out = 8'b0111_1111;
        2'b01,2'b11 : real_out = 8'b1000_0001;
      endcase

    end
    counter     = counter_var;
  end
end



endmodule
```

のシフト・レジスタの値と5番目のシフト・レジスタの値の排他的論理和(EXOR)をとり，その値が0番目のシフト・レジスタに帰還されます．これを繰り返すことによって，1クロックごとに異なるランダム・データが生成されます．

リスト1に示すように，最初にreset＝1になったとき，1ビットのcounter(カウンタ)と7ビットのshiftreg(七つのシフト・レジスタ)を初期化します．そして，resetが‘0’になると，クロックの入力に合わせてcounterの値は‘0’と‘1’を繰り返します．そして，counterが‘0’になったときのみ，シフト・レジスタの6番目の値を出力します．そのとき，シフト・レジスタの出力に同期したイネ

ーブル信号sdeを"H"('1')にして出力します．

counterの値が'1'のときは，シフト・レジスタからの出力は変化させず，イネーブル信号sdeのみを"L"('0')にして出力します．結果として，イネーブル信号sdeはクロック信号に対して1/2の周期で'0'と'1'を繰り返し出力することになります．また，このsdeの値が'1'になったとき，シフト・レジスタはランダム・データを出力することになります．

● QPSK_MAPPERブロックでQPSKのI，Q信号を生成

本ブロックでは入力される信号情報に従ってQPSK用のI信号とQ信号を出力します(**図5**)．QPSK_MAPPERブロックのVerilog HDL記述を**リスト2**に示します．

図2に示したように，QPSKでは1シンボルを(変調情報)決定するのに2ビットの情報が必要となります．そのため，ブロック内部では，QPSK_INPUTGENブロックと同じように1ビット・カウンタ(counter)を利用して，rdeが'1'となる回数をカウントします．QPSK_INPUTGENブロックのところで述べたように，rdeが2回'1'となった場合に2ビット信号が用意され，その2ビット信号に基づいてI信号とQ信号が選択され出力されるのです．

以下，処理手順を具体的に説明します．

rdeが'0'，つまり入力信号がない場合，real_outとimag_outはともに'0'を出力します．

rdeが'1'，つまり入力信号がある場合，counterの値に応じて次のような処理を行います．

- **counter＝0の場合**――入力データをレジスタ(rd_buf)に格納する．real_outとimag_outには，rdeが'0'の場合と同じように'0'を出力し続ける．そして，counterを'1'に(ビット反転)する．
- **counter＝1の場合**――rd_bufに格納された過去の1データと現在の入力データの2ビット値によるI信号とQ信号の値を求めて，real_out，imag_outとして出力する．最後に，counterを'0'に(ビット反転)する．なお，入力信号とIQ信号出力の対応は以下のようになる．

入力信号(0, 0)→IQ信号(1, 1)
入力信号(0, 1)→IQ信号(−1, 1)
入力信号(1, 0)→IQ信号(1, −1)
入力信号(1, 1)→IQ信号(−1, −1)

本ブロックで目標とする出力は**図6**に示すものです．同図では2ビット入力に応じてI信号とQ信号が正しく出力されているようすがわかります．なお，QPSK_INPUTGENブロックは2クロックごとに信号を出力します．その後段のQPSK_MAPPERブロックは2信号入力時にI信号とQ信号を出力します．つまり，I信号とQ信号の出力は2クロック×2＝4クロックごととなります．

● NQTFILTERブロックで17タップFIRフィルタを実現

次に，NQTFILTERブロックをVerilog HDLで記述します(**リスト3**)．

フィルタは，17タップFIR(finite impulse response)フィルタを用いています．演算はすべて固定長としました．タップ係数は8タップ目を中心に対称となります．そこで，積和演算では8タップ目を中心とした左右対称2タップについて被乗算信号をあらかじめ加算しておくことで，乗算器を半減させるくふうを盛り込んでいます．また，各タップ出力の和については，ビット拡張を行うことで，けたあふれを防止しています．

ナイキスト・フィルタの係数は，MATLABの「Filter Design and Analysis Tool(固定小数点フィルタの設計およびHDLコード生成機能)」を利用して計算しました．ここで計算した係数を**図7**に示します．本フィルタの特性は，DSRC(dedicated short range communication：専用狭域通信)で規定された変調信号に対する帯域制限処理に相

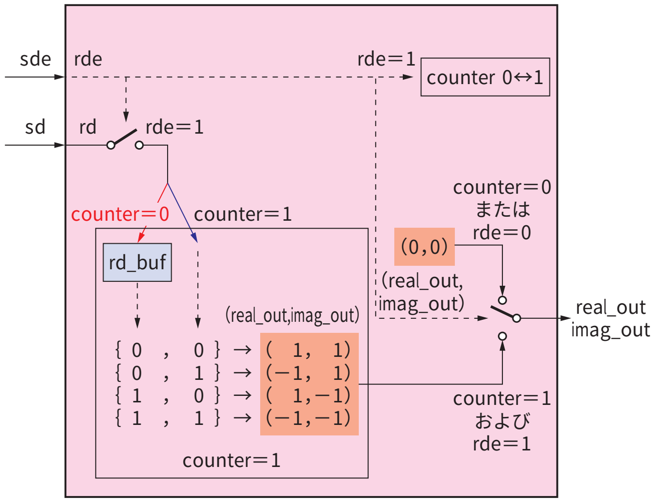


図5 QPSK_MAPPERの動作
QPSK用IQ信号を生成する．入力rdeが'1'のときに処理が行われる．counterが'0'では入力信号をrd_bufに退避させ，出力IQ信号は'0'とする．counterが'1'ではrd_bufと入力信号とで決まるI，Q信号を出力する．

3

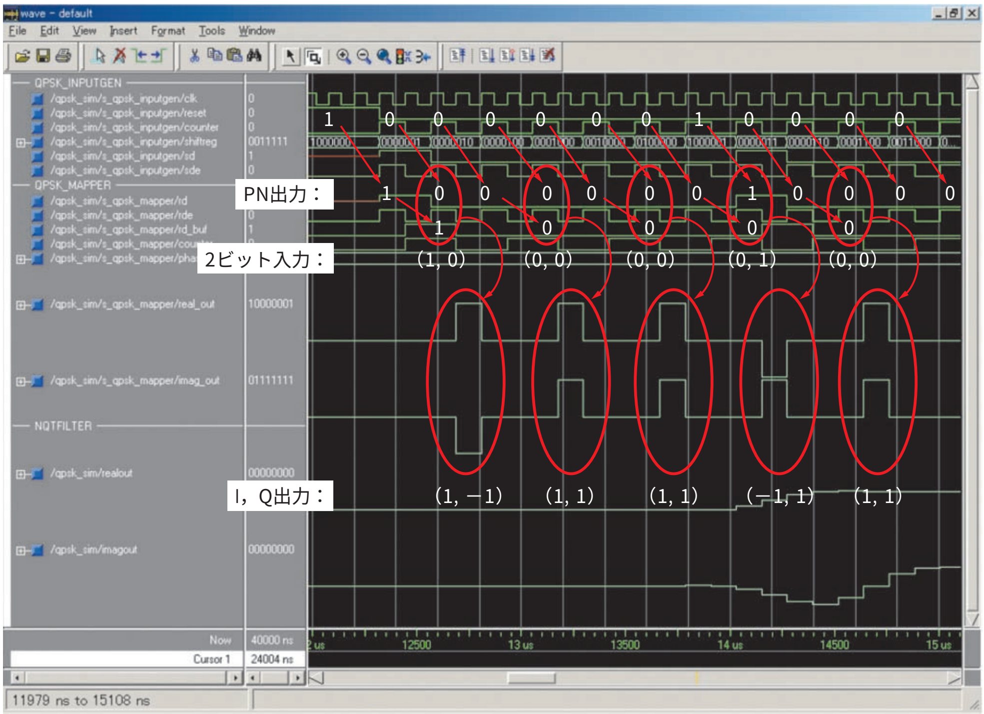


図6　期待されるシミュレーション結果

QPSK変調のシステム・レベル・シミュレーションの結果を示す．上から，入力信号発生部，*IQ*信号生成部，フィルタ処理部のそれぞれの信号を示している．入力2ビット信号ごとに*IQ*信号が生成され，それにフィルタ処理が行われているようすがわかる．

a0 = 0
a1 = 66
a2 = 0
a3 = −199
a4 = 0
a5 = 1391
a6 = 4096
a7 = 6954
a8 = 8192
a9 = 6954
a10 = 4096
a11 = 1391
a12 = 0
a13 = −199
a14 = 0
a15 = 66
a16 = 0

図7　フィルタ係数

MATLABの Filter Design & Analysis Toolで設計したサンプリング速度8.192MHz，占有帯域2.048MHzの帯域制限フィルタの係数．係数は16ビット固定小数点値で表現されている．

リスト3　NQTFILTERブロック(nqtfilter.v)

```
module nqtfilter ( clk, filter_in, filter_out );

input         clk;
input   [7:0] filter_in;
output  [7:0] filter_out;


//********** Input Signal
wire          clk;
wire    [7:0] filter_in;
//**********

//********** Output Signal
reg     [7:0] filter_out;
//**********

//********** Register
reg     [8:0] mem0, mem1, mem2, mem3, mem4, mem5, mem6,
              mem7, mem8, mem9,mem10,mem11,mem12,mem13,
                                   mem14,mem15,mem16;
reg     [8:0] mema0, mema1, mema2, mema3, mema4, mema5,
                                  mema6, mema7, mema8;
reg    [17:0] output_mem0, output_mem1, output_mem2;
reg    [20:0] output_mem_all;

reg    [20:0] output_mem_all_inv;

reg     [8:0] unsigned_in_a_0,unsigned_in_a_1,
              unsigned_in_a_2,unsigned_in_a_3,
              unsigned_in_a_4,unsigned_in_a_5,
              unsigned_in_a_6,unsigned_in_a_7,
              unsigned_in_a_8;
reg    [15:0] unsigned_in_b_0,unsigned_in_b_1,
              unsigned_in_b_2,unsigned_in_b_3,
              unsigned_in_b_4,unsigned_in_b_5,
              unsigned_in_b_6,unsigned_in_b_7,
              unsigned_in_b_8;
reg    [20:0] mpy_v_0,mpy_v_1,mpy_v_2,mpy_v_3,mpy_v_4,
              mpy_v_5,mpy_v_6,mpy_v_7,mpy_v_8;
//**********

//********** Internal Signal
reg    [15:0] output00,output01,output02,output03,
              output04,output05,output06,output07,
              output08;
wire   [15:0] in_b_u00,in_b_u01,in_b_u02,in_b_u03,
              in_b_u04,in_b_u05,in_b_u06,in_b_u07,
              in_b_u08;
//**********

//********** Parameter
assign     in_b_u00   =  16'd0;
assign     in_b_u01   =  16'd66;
assign     in_b_u02   =  16'd0;
assign     in_b_u03   = -16'd199;
assign     in_b_u04   =  16'd0;
assign     in_b_u05   =  16'd1391;
assign     in_b_u06   =  16'd4096;
assign     in_b_u07   =  16'd6954;
assign     in_b_u08   =  16'd8192;
//**********

always @(posedge clk) begin
  if (mema0[8]==1'b0)      unsigned_in_a_0  =  mema0;
  else                     unsigned_in_a_0  =
                                  ~(mema0 - 9'b1);
  if (in_b_u00[15]==1'b0) unsigned_in_b_0  =  in_b_u00;
  else                     unsigned_in_b_0  =
                              ~(in_b_u00 - 16'b1);
  mpy_v_0 = unsigned_in_a_0 * (unsigned_in_b_0[15:4]);
  if (mema0[8]^in_b_u00[15] == 1'b1) mpy_v_0 =
                                 ~mpy_v_0 + 21'b1;
  output00 = {mpy_v_0[20], mpy_v_0[16:2]};
end

always @(posedge clk) begin
```

```
  if (mema1[8]==1'b0)      unsigned_in_a_1  =  mema1;
  else                     unsigned_in_a_1  =
                                          ~(mema1 - 9'b1);
  if (in_b_u01[15]==1'b0) unsigned_in_b_1  =  in_b_u01;
  else                     unsigned_in_b_1  =
                                      ~(in_b_u01 - 16'b1);
  mpy_v_1 = unsigned_in_a_1 * (unsigned_in_b_1[15:4]);
  if (mema1[8]^in_b_u01[15] == 1'b1) mpy_v_1 =
                                         ~mpy_v_1 + 21'b1;
  output01 = {mpy_v_1[20], mpy_v_1[16:2]};
end

always @(posedge clk) begin
  if (mema2[8]==1'b0)      unsigned_in_a_2  =  mema2;
  else                     unsigned_in_a_2  =
                                          ~(mema2 - 9'b1);
  if (in_b_u02[15]==1'b0) unsigned_in_b_2  =  in_b_u02;
  else                     unsigned_in_b_2  =
                                      ~(in_b_u02 - 16'b1);
  mpy_v_2 = unsigned_in_a_2 * (unsigned_in_b_2[15:4]);
  if (mema2[8]^in_b_u02[15] == 1'b1) mpy_v_2 =
                                         ~mpy_v_2 + 21'b1;
  output02 = {mpy_v_2[20], mpy_v_2[16:2]};
end

always @(posedge clk) begin
  if (mema3[8]==1'b0)      unsigned_in_a_3  =  mema3;
  else                     unsigned_in_a_3  =
                                          ~(mema3 - 9'b1);
  if (in_b_u03[15]==1'b0) unsigned_in_b_3  =  in_b_u03;
  else                     unsigned_in_b_3  =
                                      ~(in_b_u03 - 16'b1);
  mpy_v_3 = unsigned_in_a_3 * (unsigned_in_b_3[15:4]);
  if (mema3[8]^in_b_u03[15] == 1'b1) mpy_v_3 =
                                         ~mpy_v_3 + 21'b1;
  output03 = {mpy_v_3[20], mpy_v_3[16:2]};
end

always @(posedge clk) begin
  if (mema4[8]==1'b0)      unsigned_in_a_4  =  mema4;
  else                     unsigned_in_a_4  =
                                          ~(mema4 - 9'b1);
  if (in_b_u04[15]==1'b0) unsigned_in_b_4  =  in_b_u04;
  else                     unsigned_in_b_4  =
                                      ~(in_b_u04 - 16'b1);
  mpy_v_4 = unsigned_in_a_4 * (unsigned_in_b_4[15:4]);
  if (mema4[8]^in_b_u04[15] == 1'b1) mpy_v_4 =
                                         ~mpy_v_4 + 21'b1;
  output04 = {mpy_v_4[20], mpy_v_4[16:2]};
end

always @(posedge clk) begin
  if (mema5[8]==1'b0)      unsigned_in_a_5  =  mema5;
  else                     unsigned_in_a_5  =
                                          ~(mema5 - 9'b1);
  if (in_b_u05[15]==1'b0) unsigned_in_b_5  =  in_b_u05;
  else                     unsigned_in_b_5  =
                                      ~(in_b_u05 - 16'b1);
  mpy_v_5 = unsigned_in_a_5 * (unsigned_in_b_5[15:4]);
  if (mema5[8]^in_b_u05[15] == 1'b1) mpy_v_5 =
                                         ~mpy_v_5 + 21'b1;
  output05 = {mpy_v_5[20], mpy_v_5[16:2]};
end

always @(posedge clk) begin
  if (mema6[8]==1'b0)      unsigned_in_a_6  =  mema6;
  else                     unsigned_in_a_6  =
                                          ~(mema6 - 9'b1);
  if (in_b_u06[15]==1'b0) unsigned_in_b_6  =  in_b_u06;
  else                     unsigned_in_b_6  =
                                      ~(in_b_u06 - 16'b1);
  mpy_v_6 = unsigned_in_a_6 * (unsigned_in_b_6[15:4]);
  if (mema6[8]^in_b_u06[15] == 1'b1) mpy_v_6 =
                                         ~mpy_v_6 + 21'b1;
  output06 = {mpy_v_6[20], mpy_v_6[16:2]};
end

always @(posedge clk) begin
  if (mema7[8]==1'b0)      unsigned_in_a_7  =  mema7;
  else                     unsigned_in_a_7  =
                                          ~(mema7 - 9'b1);
  if (in_b_u07[15]==1'b0) unsigned_in_b_7  =  in_b_u07;
  else                     unsigned_in_b_7  =
                                      ~(in_b_u07 - 16'b1);
  mpy_v_7 = unsigned_in_a_7 * (unsigned_in_b_7[15:4]);
  if (mema7[8]^in_b_u07[15] == 1'b1) mpy_v_7 =
                                         ~mpy_v_7 + 21'b1;
  output07 = {mpy_v_7[20], mpy_v_7[16:2]};
end

always @(posedge clk) begin
  if (mema8[8]==1'b0)      unsigned_in_a_8  =  mema8;
  else                     unsigned_in_a_8  =
                                          ~(mema8 - 9'b1);
  if (in_b_u08[15]==1'b0) unsigned_in_b_8  =  in_b_u08;
  else                     unsigned_in_b_8  =
                                      ~(in_b_u08 - 16'b1);
  mpy_v_8 = unsigned_in_a_8 * (unsigned_in_b_8[15:4]);
  if (mema8[8]^in_b_u08[15] == 1'b1) mpy_v_8 =
                                         ~mpy_v_8 + 21'b1;
  output08 = {mpy_v_8[20], mpy_v_8[16:2]};
end


always @(posedge clk) begin
  output_mem0 = {output00[15],output00[15],output00}
               +{output01[15],output01[15],output01}
               +{output02[15],output02[15],output02}
               +{output03[15],output03[15],output03};
  output_mem1 = {output04[15],output04[15],output04}
               +{output05[15],output05[15],output05}
               +{output06[15],output06[15],output06}
               +{output07[15],output07[15],output07};
  output_mem2 = {output08[15],output08[15],output08};
end

always @(posedge clk) begin
  output_mem_all = {output_mem0[17],output_mem0[17],
                      output_mem0[17],output_mem0}
                  +{output_mem1[17],output_mem1[17],
                      output_mem1[17],output_mem1}
                  +{output_mem2[17],output_mem2[17],
                     output_mem2[17],output_mem2};
end

always @(posedge clk) begin

  mem16=mem15;
  mem15=mem14; mem14=mem13; mem13=mem12;
                              mem12=mem11; mem11=mem10;
  mem10=mem9;  mem9 =mem8;  mem8 =mem7;
                              mem7 =mem6;  mem6 =mem5;
  mem5 =mem4;  mem4 =mem3;  mem3 =mem2;
                              mem2 =mem1;  mem1 =mem0;

  mem0 = {filter_in[7], filter_in};
end

always @(posedge clk) begin
  mema0 =mem0 + mem16;  mema1 =mem1
      + mem15; mema2 =mem2 + mem14; mema3 =mem3 + mem13;
  mema4 =mem4 + mem12;  mema5 =mem5
       + mem11; mema6 =mem6 + mem10; mema7 =mem7 + mem9;
  mema8 =mem8;
end

always @(posedge clk) begin

  output_mem_all_inv = ~output_mem_all + 1'b1;

  if      (output_mem_all[20]==
                1'b0 && output_mem_all > 32767) begin
    filter_out =  127;
  end
  else if (output_mem_all[20]==1'b1 && output_mem
                               _all_inv > 32768 ) begin
    filter_out = -128;
  end
  else begin
    filter_out = {output_mem_all[20],
                                 output_mem_all[14:8]};
  end

end

endmodule
```

3

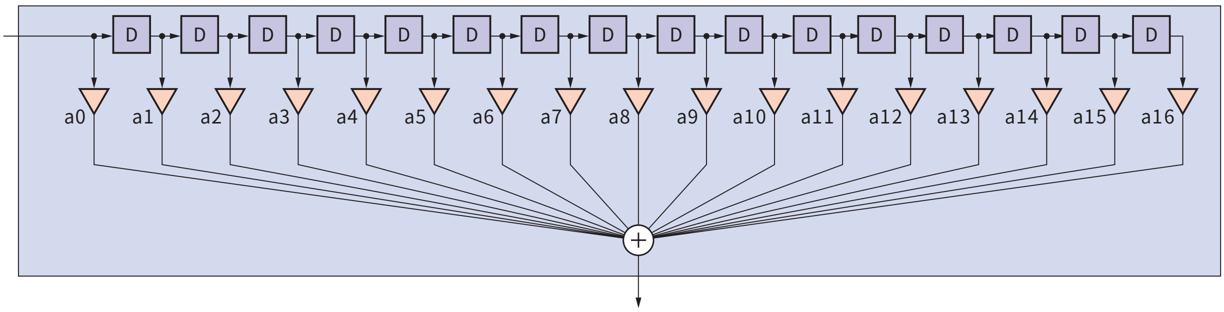


図8　一般的なFIRフィルタの構成

四角形で示される遅延線と三角形で示されるタップ係数との乗算処理，また各タップ乗算処理の結果の総和をとる加算器から構成される．本構成の場合，加算器計算の処理負荷が高く，動作速度が低下してしまう．

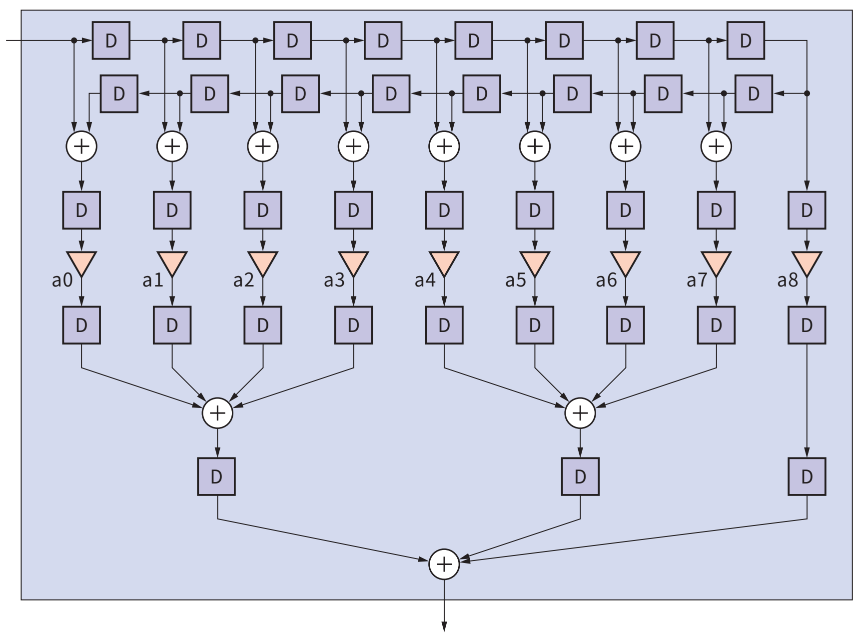


図9
今回設計したナイキスト・フィルタの構成

今回設計したフィルタ構成では，タップ係数が対称であることを利用して，同じ係数との乗算を行うことになる信号をあらかじめ加算しておくことで，乗算器を半減させている．また，各タップ出力の加算は，その加算数の最大値を4とすることで処理遅延を低減させ動作速度を高めている．

当します．占有帯域を2.048MHzとし，4倍オーバサンプリングを行い，サンプリング速度を8.192MHzと想定して設計しました．なお，今回設計したフィルタのインパルス応答については，**図3**を参照してください．

FIRフィルタの一般的な構造を**図8**に示します．図のように，信号を1クロック遅らせる遅延素子，加算器，乗算器を利用して，タップ係数との畳み込み演算を行います．これに対して，今回HDL化したFIRフィルタの構成は**図9**のようになります．ここでは，タップ計算の処理やその中に含まれる乗算器を半減させたり，加算器入力数を最大4個とするなど，処理遅延を減らすくふうを盛り込んでいます．

各タップにおける計算処理の構成を**図10**に示します．タップ計算用の乗算処理では，ナイキスト・フィルタ・ブロックから入力されるタップ入力値とタップ係数を乗算する処理を行います．今回の設計ではタップ係数値は16ビットで入力されますが，そのうちの上位12ビットのみを利用して乗算を行うことにしました．

VHDLでは，符号付き演算用ライブラリ（IEEE.std_logic_signed）を利用することで，乗算処理における符号付き計算を行うことができます．一方，Verilog HDLでは符号付き演算はサポートされていないため，符号についてはユーザ側で回路を構成する必要があります．そこで今回は，符号検出による絶対値化された2値に対する乗算を行い，再び符号を付け直すといった処理を記述しました．

また，タップ出力の総和を計算した後，出力信号8ビットの切り出しを行う際には，上限と下限のクリッピング処

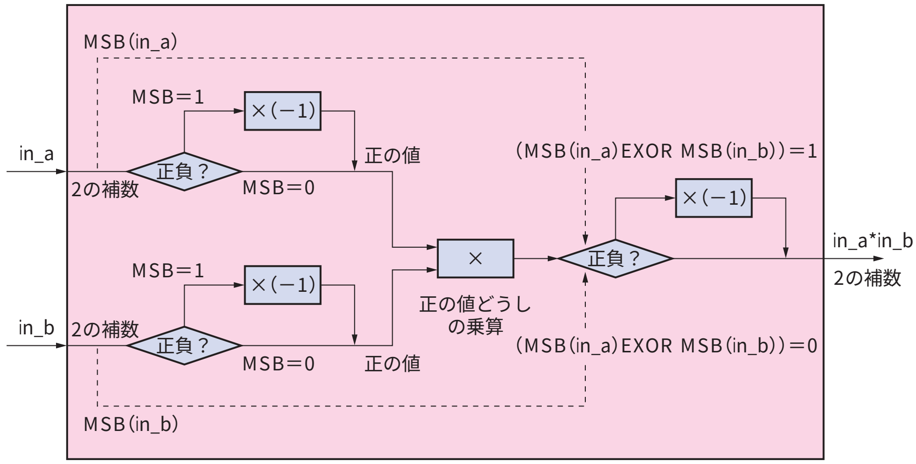


図10
タップの計算(2の補数どうしの乗算)
2の補数どうしの乗算を行うため，まず入力2信号の絶対値を取り，絶対値の積を計算し，最後に2信号入力の最上位ビット(符号ビット)の排他的論理和をとることで積の符号を調べている．負の場合にはマイナス値にして出力する．

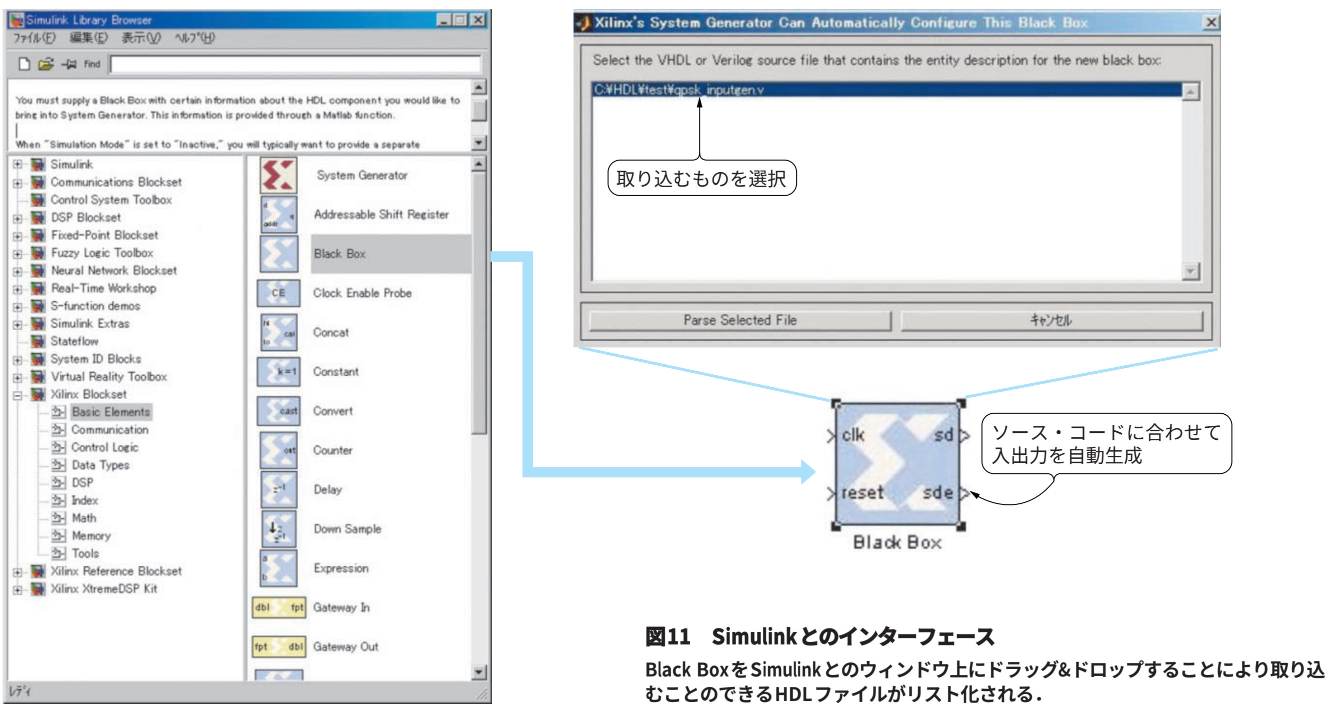


図11　Simulinkとのインターフェース
Black BoxをSimulinkとのウィンドウ上にドラッグ&ドロップすることにより取り込むことのできるHDLファイルがリスト化される．

理を行い，けたあふれを防止しています．

以上のようにして，帯域制限特性を備えた係数を持たせることで，4倍オーバサンプリング処理のための17タップFIRフィルタを実現しました．

3 ソース・コードをSimulinkへロードする

ここまでの説明で，QPSKを用いて送信機を実現するための三つのブロックをHDLで記述しました(ファイル名はqpsk_inputgen.v，qpsk_mapper.v，nqtfilter.v)．これらを一つのフォルダに保存します．

次に，これらの各ブロックの動作検証を行います．通常，動作検証にはHDLシミュレータを用いますが，ここではThe MathWorks社の信号処理用システム・レベル・シミュレータ「MATLAB/Simulink」，およびXilinx社の「System Generator for DSP」を用います．

まず，Simulinkを起動します．すると，「Simulink Library Browser」の画面が表示されます(図11)．メニュー・バーの「ファイル」→「新規作成」を選択し，上記三つ

のプログラムを格納しているフォルダにSimulinkのファイルを作ります(名まえは何でもよい). System Generator for DSPが正常にインストールされていれば, 「Simulink Library Browser」の画面にある機能ツリーの中に「Xilinx Blockset」という項目が現れるはずです. 「Xilinx Blockset」を展開し, 「Basic Elements」の「Black Box」というブロックを新規に作成したSimulinkファイルの上に移動します. すると, **図11**の右上のようなウィンドウが自動的に開きます.

そこで, 同一フォルダ内にあるHDLで書かれた組み込み可能なファイル名が示されるので, 必要となるファイルを選択します(例えば, **図11**ではqpsk_inputgen.vを選択している). すると, 自動的にそのファイルを解析して, **図11**の右下のような入出力を持ったモジュールが生成されます. また, HDLで記述されたファイルとSimulinkのインターフェースとなるqpsk_inputgen_configmというファイルが生成されます. qpsk_mapper.v, nqtfilter.vについても同じ手順でモジュールを作成します.

このとき, 生成したモジュールとHDLソース・コードの入出力ポート名が一致していることを確認してください. また, 各モジュールをダブルクリックし, 「Simulation Mode」というパラメータについてプルダウン・メニューから「Use HDL Co-simulation」を選択します. 「HDL Co-Simulator To Use」というパラメータには「ModelSim」と入力してください.

4 評価用モジュールを作ってシミュレーション

上記の作業によってVerilog HDLのソース・コードをSimulinkに取り込み, モジュール化しました. 次に, 各ブロックにSimulinkのモジュールを加え, Simulink上で動作する評価モデルを作ります.

● qpsk_inputgenモジュールを評価

図12に, qpsk_inputgen.vの評価用Simulinkモデルを示します. 評価にあたっては, 次の四つの事項をかならず実施する必要があります.

まず, 「Xilinx Blockset」の中にあるSystem Generatorモジュール(**図12**のA)を評価用Simulinkモデルの上にドラッグ・アンド・ドロップします. 同じように, ModelSimモジュール(**図12**のB)をドラッグ・アンド・ドロップします. こうすることで, ModelSimとSimulinkの間で協調シミュレーションが行えるようになります.

次に, fptとdblとの変換ゲートウェイ(**図12**のC)と変換モジュール(**図12**のD)を評価用Simulinkモデルの上にドラッグ・アンド・ドロップします. qpsk_inputgenモジュールは, 「Xilinx Blockset」の中にあるモジュール以外とは直接接続することができません. qpsk_inputgenモジ

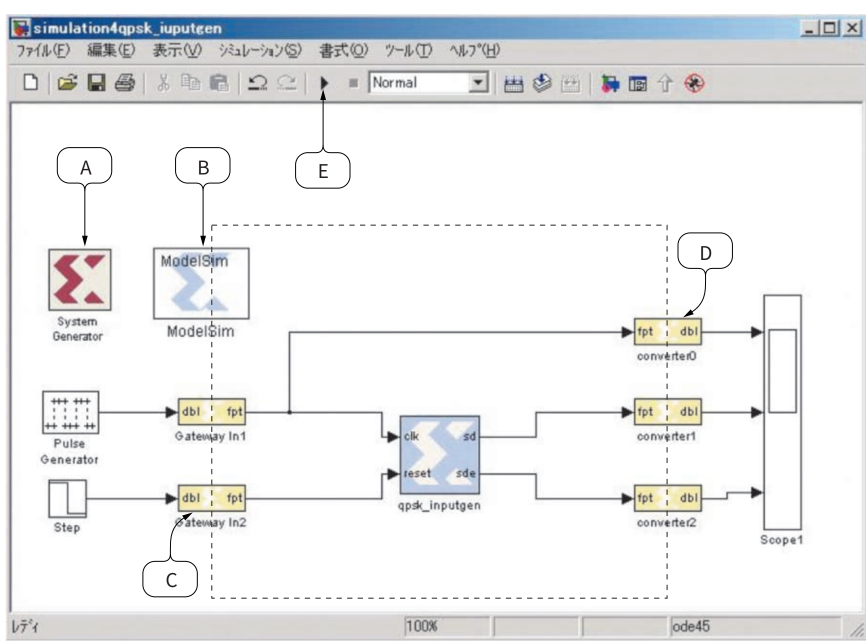


図12
qpsk_inputgenの評価モデル
QPSKの信号生成器の評価を行う点線間は固定小数点で計算される.

ュールはあくまでも固定小数点(fixed point)で計算されます．

今回のシミュレーションはModelSimによって行います．また，それ以外の部分はSimulinkによって倍精度(double)で計算されるため，fptとdblとの変換ゲートウェイ(Gateway)や変換モジュール(converter)が必要になります(Gatewayやconverterは「Xilinx Blockset」の中に用意されている)．図12では，点線の内部が固定小数点で計算されます．

最後に，各ポートに入力する信号をSimulinkの基本モジュールを用いて供給します．図12では，クロックとしてPulse Generatorを，reset信号としてステップ関数生成モジュール(Step)を用います．また，qpsk_inputgenモジュールから出力されるsd，sdeおよび入力クロックを倍精度に変換して，Simulinkの基本モジュールであるScopeモジュールに出力します．図12のEに示すボタンを押すと，ModelSimが自動的に立ち上がり，SimulinkとModelSimの協調シミュレーションが始まります．

図13にモジュールの出力を示します．上からクロック，ランダム・データsd，sdに同期したイネーブル信号sdeを示しています．QPSK_INPUTGENブロックで設計したとおり，クロックの1/2の周期でsdeが'0'と'1'を繰り返し出力しています．また，sdeに同期してランダム・データが出力されていることがわかります．

● qpsk_mapperモジュールを評価

同じようにしてqpsk_mapperモジュールの評価を行います．図14に評価用Simulinkモデルを示します．基本的に

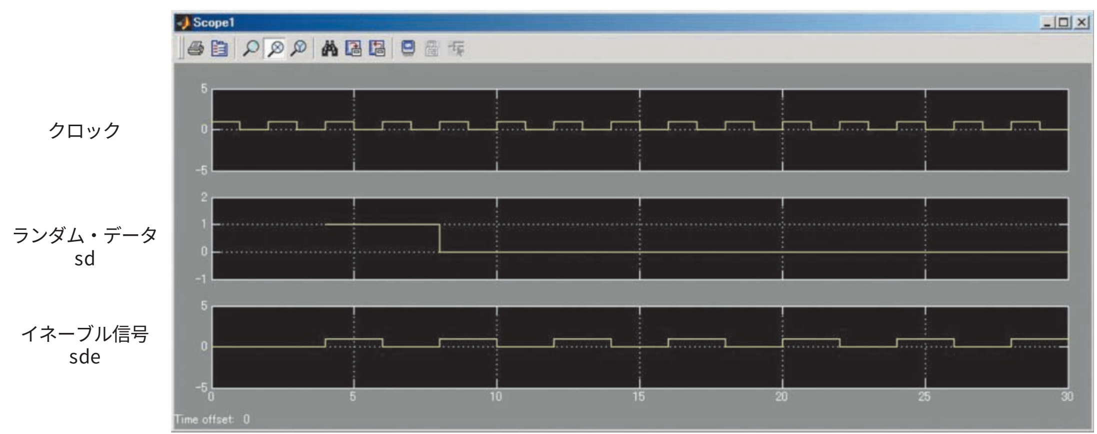


図13 qpsk_inputgenの評価結果
イネーブル信号に合わせて図4の回路どおりにデータが出力されているようすがわかる．

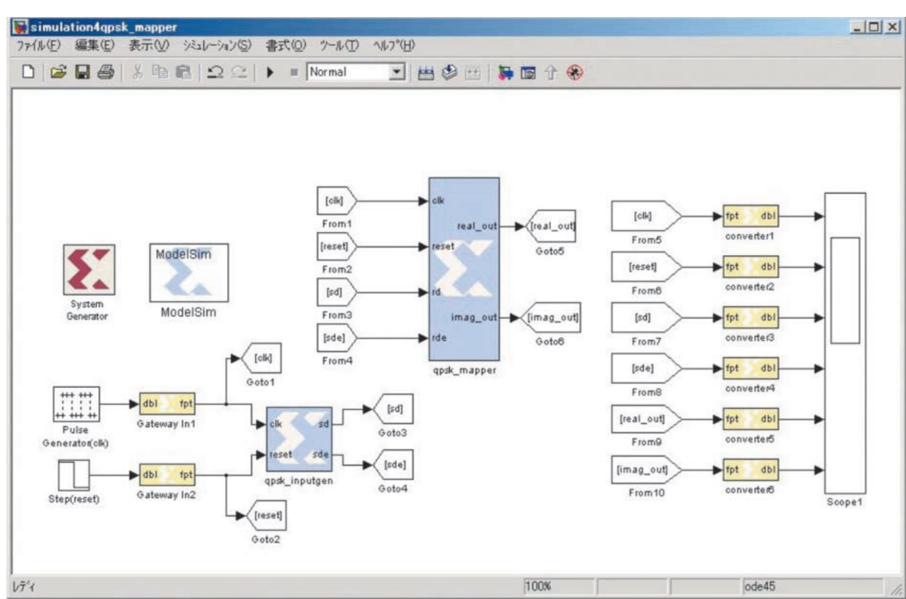


図14 qpsk_mapperの評価モデル
qpsk_mapperを評価するプログラム．Qpsk_inputgenブロックの出力を入力し動作する．

は，図12で生成したランダム・データrd(sd)とそのイネーブル信号rde(sde)，クロック，リセット信号をqpsk_mapperモジュールに入力するだけです．結果として，入力信号を2ビットずつ取り込み，図2に合わせてI信号real_outとQ信号imag_outを出力します．

図15にその評価結果を示します．クロック信号clk，リセット信号reset，ランダム・データrd，イネーブル信号rde，I信号real_out，Q信号imag_outを出力しています．rdが1，0，0，0，…と変化し，2ビットずつ(1，0)，(0，0)，…に区切られ，その結果，図2で示したように(1，−1)，(1，1)，…と出力されていることがわかります．また，フィルタを入れることを前提に，各出力は4クロックごとに出力され，その間は‘0’を出力しています．

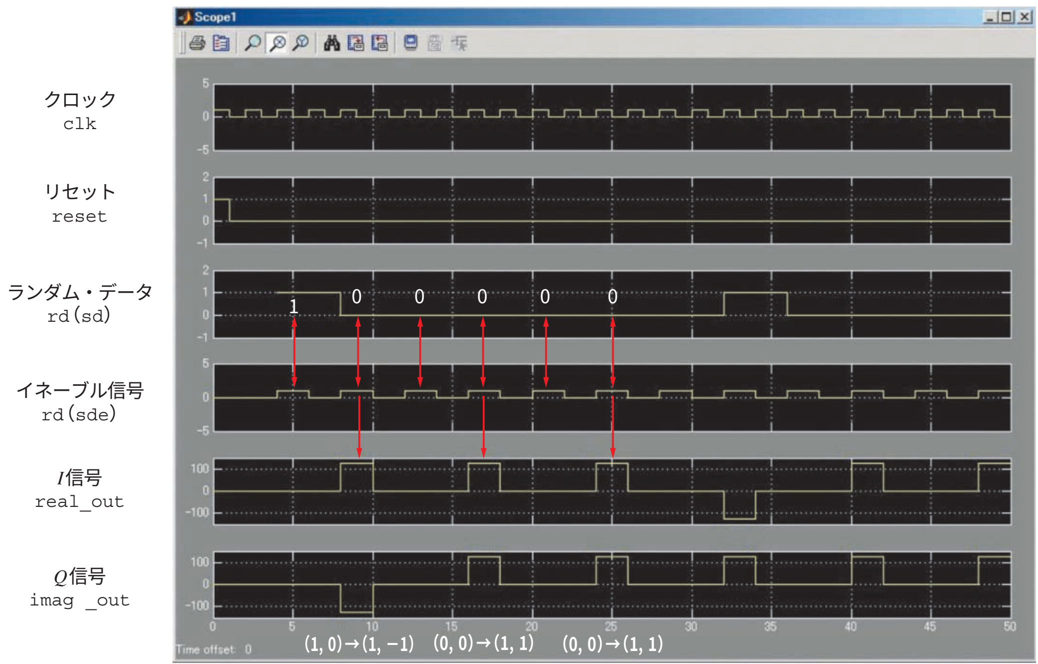


図15
qpsk_mapperの評価結果
qpsk_mapperの出力は図6で期待された結果が出力されている．

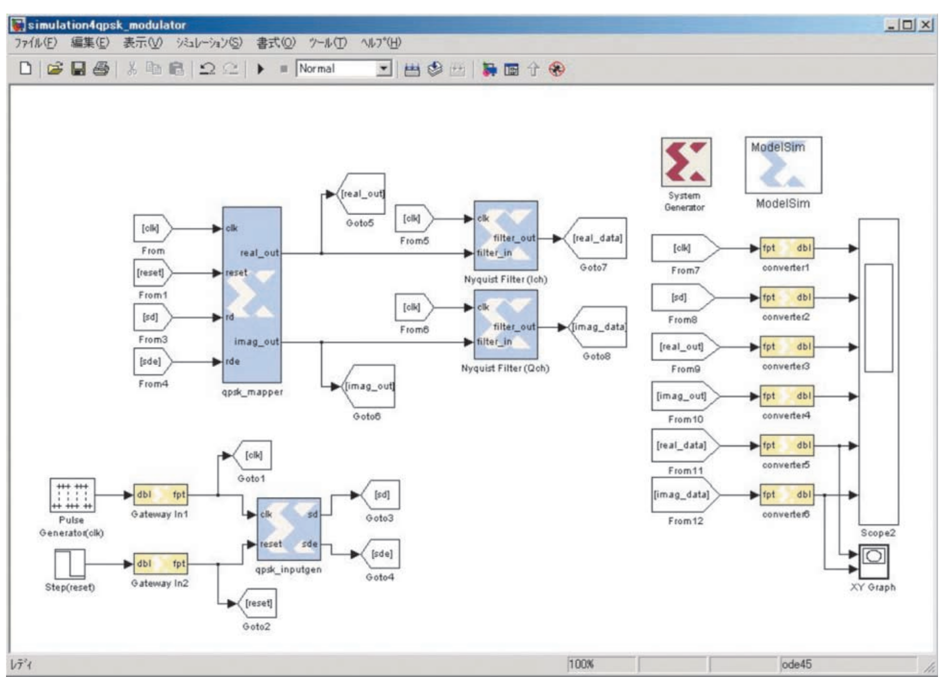


図16
QPSK送信機の評価モデル
qpsk_mapper出力をナイキスト・フィルタ・ブロックに接続し，QPSK送信機が完成する．

● nqtfilterモジュールを評価

最後にnqtfilterモジュールを加えて，QPSK送信機としての評価を行います．

図16に評価用Simulinkモデルを示します．基本的には図14で生成したreal_outとimag_outをそれぞれナイキスト・フィルタであるnqtfilterモジュールに入力することによって，出力としてフィルタリングされた*I*信号(mod_real_out)と*Q*信号(mod_imag_out)が得られます．そして，SimulinkのScope画面に上からclk，sd，real_out，imag_out，mod_real_out，mod_imag_outを出力します(図17)．

出力信号がほんとうにQPSK信号であるかどうかを確認するため，Simulinkの基本モジュールであるXY Graphモジュール(*I*信号を横軸に，*Q*信号を縦軸にとった信号空間ダイヤグラム表示機能)を利用して，出力するmod_real_outとmod_imag_outを*X*軸と*Y*軸にマッピングします．QPSK変調を利用した送信機としてきちんと動作している場合は，本特集 第2章，p.55の図24に示した信号点パターンが出力されるはずです(図18)．このように，HDLのシミュレーションを行いながら，システム・レベル・シミュレーションの良いところを利用できるのが協調シミュレーションの特徴です．

5 合成・配置配線用プロジェクト・ファイルを作成

以上で，QPSKを用いた送信機の設計が終了しました．最終的に，図19に示すように評価モデルを必要最小限に抑えました．

図19において，点線で囲った部分を論理合成，配置配線できるファイルに自動的に変換したいという方もいると思います．例えば，Xilinx社のISEシリーズのような論理合成，配置配線ツール用のファイルに変換したいということであれば，System Generator for DSPを用います．図19のSystem Generatorモジュールをクリックすると，図20に示すメニューが出ます．そこにある各種パラメータに必要な情報を入力し，［Generate］ボタンを押すと，ISEシリーズ用のプロジェクト・ファイルが生成されます．

今回は本誌2005年1月号の付属FPGA（Xilinx社製Spartan-3）基板を用いたり，クロックとして8.192MHzを使うことを考えると，図20のようなパラメータ設定になります．この結果，生成されたプロジェクト・ファイルをISEシリーズに読み込ませることで，論理合成や配置配線が行えるようになります(詳細は本特集の第5章で説明する)．

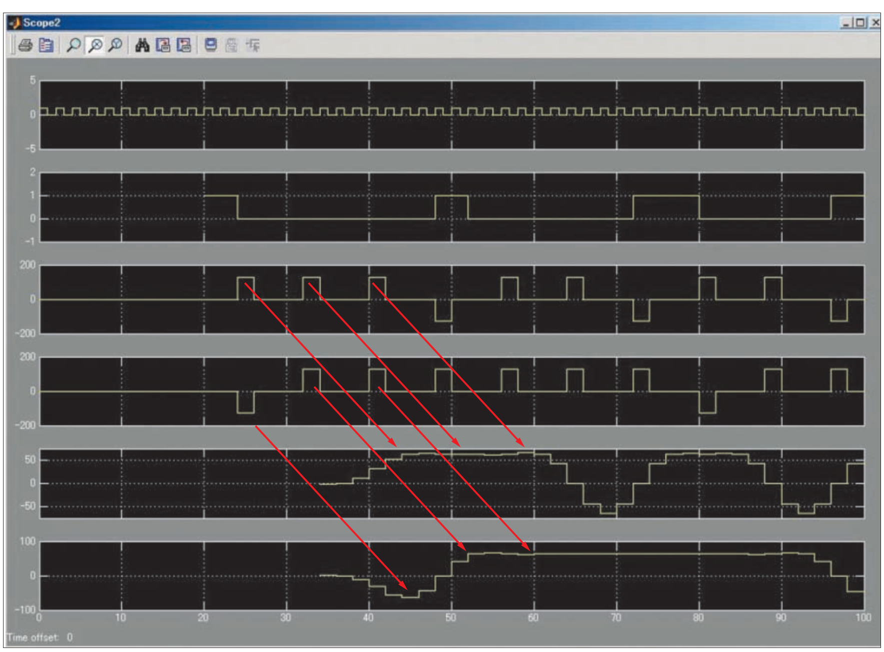


図17 QPSK送信機の評価結果
出力の表示にSimulinkを用いることにより，アナログ値の視覚的によい評価が容易に行うことができる．

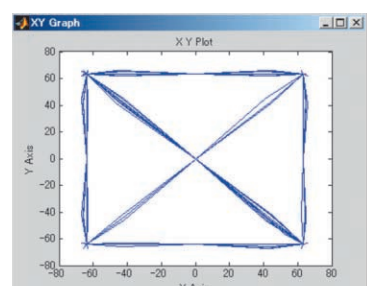


図18 XY Graphによる信号点の表示
XY Graphモジュールの出力が，QPSKの変調信号の生成に成功していることがわかる．

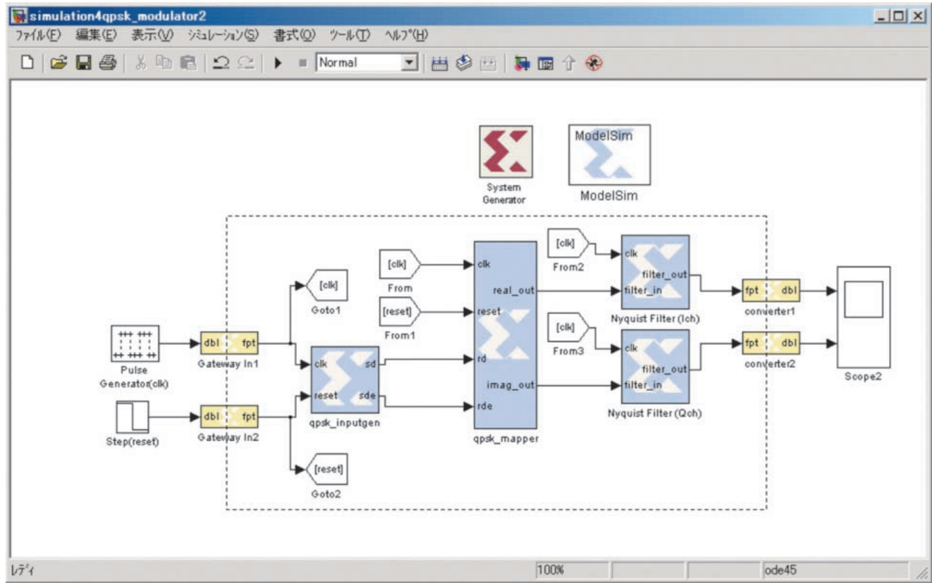


図19
図16のモデルを必要最小限に削減したモデル

図16から観測用のブロックを外し，論理合成・配置配線用プロジェクト･ファイルの作成準備をする．

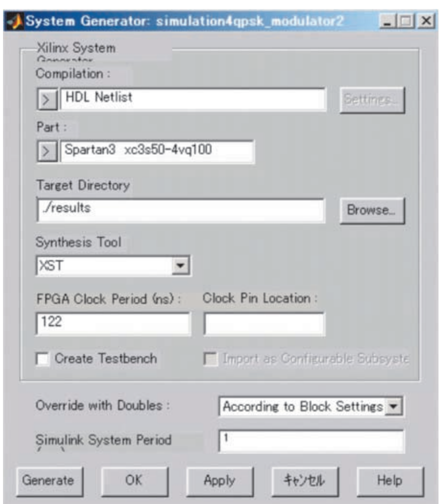


図20　System Generatorブロックの設定条件

ここを使用するFPGAに従ってカスタマイズかる．

＊　　＊　　＊

ここでは，QPSKを用いたソフトウェア無線機（送信機の部分）のFPGA用回路データを作成しました．この回路データは，本誌の付属DVD-ROMに収録されています．

このままだと単なる一つのQPSK用の送信機しか構成できません．ソフトウェア無線機なので，複数の送信機を切り替えられるようにする必要があります．そこで第4章では，携帯電話システム，高度道路交通システム用専用狭域通信システム（DSRC）などに用いられているπ/4シフトQPSK変調方式とASK変調方式を切り替えて送信できる送信機を作成します．この一連の作業によって，ソフトウェア無線機の開発を体感していただければと思います．

はらだ・ひろし
独立行政法人 情報通信研究機構
よこやま・あきひさ
（株）トヨタIT開発センター

<筆者プロフィール>

原田博司．旧郵政省通信総合研究所入所以来，ディジタル信号処理を利用した無線通信システム，ソフトウェア無線技術に関する研究に従事．現在，独立行政法人 情報通信研究機構 横須賀無線通信研究センター ワイヤレスアクセスグループグループリーダー．

横山明久．2001年よりトヨタIT開発センターにてソフトウェア無線技術を用いたテレマティクスの研究開発に従事．また，2002年より情報通信研究機構 ワイヤレスアクセスグループ特別研究員を併任．ソフトウェア無線機の効率的実現のための信号処理プラットホーム技術の研究開発に従事．